A Bag of Useful Tricks for Practical Neural Machine Translation: Embedding Layer Initialization and Large Batch Size

Masato Neishi* and **Jin Sakuma*** and **Satoshi Tohda*** and **Shonosuke Ishiwatari**
The University of Tokyo
{neishi, jsakuma, tohda, ishiwatari}@tkl.iis.u-tokyo.ac.jp

Naoki Yoshinaga and **Masashi Toyoda**
Institute of Industrial Science, the University of Tokyo
{ynaga, toyoda}@iis.u-tokyo.ac.jp

Abstract

In this paper, we describe the team UT-IIS's system and results for the WAT 2017 translation tasks. We further investigated several tricks including a novel technique for initializing embedding layers using only the parallel corpus, which increased the BLEU score by 1.28, found a practical large batch size of 256, and gained insights regarding hyperparameter settings. Ultimately, our system obtained a better result than the state-of-the-art system of WAT 2016. Our code is available on https://github.com/nem6ishi/wat17.

1 Introduction

The advent of neural networks in machine translation has contributed greatly to the translation quality. Since proposed in (Cho et al., 2014; Sutskever et al., 2014), the sequence-to-sequence (SEQ2SEQ) model has been achieving the state-of-the-art performance when combined with the attention mechanism (Bahdanau et al., 2015). Many studies have focused on modifying the SEQ2SEQ network structure, including modifying the encoder (Eriguchi et al., 2016; Gehring et al., 2017; Li et al., 2017; Chen et al., 2017), or the decoder (Ishiwatari et al., 2017; Eriguchi et al., 2017; Aharoni and Goldberg, 2017; Wu et al., 2017).

While these network structure modifications have been found to improve the translation quality, many systems, including the best system from WAT 2016 (Cromieres et al., 2016), still depend on the vanilla SEQ2SEQ model, the model with the attention mechanism. Denkowski and Neubig (2017) confirmed the large impact of common techniques such as training algorithms, subwords (Sennrich et al., 2016) and model ensembles upon this vanilla SEQ2SEQ model. This suggests that there may be some unexplored tricks we may apply to the vanilla model to significantly improve the translation quality.

This paper describes the system that we have built for the ASPEC (Nakazawa et al., 2016) en-ja translation subtask for WAT 2017 (Nakazawa et al., 2017), which incorporates a novel trick, embedding layer initialization. This trick improves upon the vanilla SEQ2SEQ model by initializing the word embedding layers of both the encoder and the decoder with word embeddings that are pretrained on the parallel corpus. Our system involves generating multiple models using SEQ2SEQ with embedding layer initialization, exhaustively searching for a combination of models with the highest ensemble score, and finally, conducting a beam search on the best ensemble. We achieved a BLEU score of 38.93 on the ASPEC en-ja translation task as the team UT-IIS, which outperforms the state-of-the-art system of WAT 2016.

Furthermore, we have provided insight on NMT by detailing experiments on the tricks used in our system. This includes testing embedding layer initialization with multiple word embedding methods (§ 5.3.1), a thorough investigation of the point where increasing the batch size ceases to be beneficial (§ 5.3.2), finding the optimal learning rate (§ 5.3.3), and investigating the relation between the number of models used in the ensemble and translation performance (§ 5.3.4). We believe that these findings, particularly regarding embedding layer initialization and practical batch size, can serve as useful tricks for future neural machine translation (NMT) systems.

The structure of this paper is as follows. In § 2, we review related work, and in § 3, we present an overview of NMT. We describe our system in § 4 and show the official evaluation result and further investigations in § 5. We conclude our work in § 6.

*Authors contributed equally.

Proceedings of the 4th Workshop on Asian Translation, pages 99–109,
Taipei, Taiwan, November 27, 2017.

2 Related Work

In this section, we will survey existing techniques used in NMT systems. We first focus on pretraining, for which we have proposed a new method, and then batch size, of which we have confirmed the effect.

2.1 Pretraining

Training deep neural networks with a relatively small amount of training data risks creating a model that performs poorly. One technique used to minimize this drawback is pretraining of the model (Hinton et al., 2006; Bengio et al., 2007), which initializes (part of) the parameters of the model using parameters of another model.

Pretraining has led to promising results in NLP tasks using SEQ2SEQ models. In languages with a small amount of supervised data, it has been found that NMT results can be improved by transferring parameters from a high-resource language pair to a low-resource one (Zoph et al., 2016). Gülçehre et al. (2015) proposed a method using a combination of the output probabilities of a language model trained on large monolingual corpora and a SEQ2SEQ NMT model, which are both trained separately. Venugopalan et al. (2016) studied different types of systems combined with a language model under the video description generation task and also introduced a method to initialize the embedding layer and the RNN layer of the decoder of the SEQ2SEQ based model with pretrained parameters of the language model. They additionally proposed a method to initialize the embedding layer of the decoder with pretrained GloVe (Pennington et al., 2014) embeddings. Ramachandran et al. (2017) initializes both the encoder and decoder of the SEQ2SEQ model with attention using language models trained on monolingual, unlabeled corpus of the source and target domains, respectively. This led to a significant improvement over the baseline.

The aforementioned studies, however, demand a large computational cost for pretraining a complex language model on large external data. Although Ramachandran et al. (2017) has provided a comparison of a system initialized using a language model trained only on the parallel corpus (in addition to their proposed method) to a baseline system without initialization, the translation performance did not improve but rather degraded with this setting.

Our work investigates the effect of initializing only the embedding layer using embeddings pretrained at low cost from the parallel corpus. We will later confirm that this initialization leads to a BLEU score increase of 1.28 (§ 5.3.1).

2.2 Batch Size

Batch size is the number of data points in a mini-batch, which is a representative portion of the training data from which the gradient is calculated at each step in the stochastic gradient descent (SGD) optimizer (or its variants). In general, the batch size chosen for deep neural networks ranges from 32 to 512. It is known that a batch size that is too large leads to performance degradation in deep neural networks (Keskar et al., 2017).

Recent studies in NMT have used values such as 64 (Rush et al., 2015) or 128 (Wu et al., 2016). While Britz et al. (2017) conducted a thorough investigation of hyperparameters in NMT, they fixed batch size to 128. The specific effect of batch size on NMT was studied by Morishita et al. (2017), who found that, for batch sizes of 8 to 64, a larger batch size has a positive impact on model performance.

In this study, we seek to empirically clarify the point where increasing the batch size no longer improves NMT performance. Our work expands upon Morishita et al. (2017) and further investigates how NMT performance varies with larger batch sizes, up to 512.

3 The Vanilla SEQ2SEQ Model

The SEQ2SEQ (or encoder-decoder) model have been achieving the state-of-the-art in machine translation (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). Bahdanau et al. (2015) further improved this model by proposing the attention mechanism.

This neural machine translation (NMT) approach involves an RNN-based encoder that converts the source sentence into vector representations which are then converted into the output sentence by an RNN-based decoder.

While there are several variations in encoder implementation, including long-short term memory (LSTM) (Hochreiter and Schmidhuber, 1997), gated recurrent unit (GRU) (Cho et al., 2014), and convolutional neural network (CNN) (Gehring et al., 2017), our system implements a two-layer bidirectional LSTM for the encoder. A bidirec-

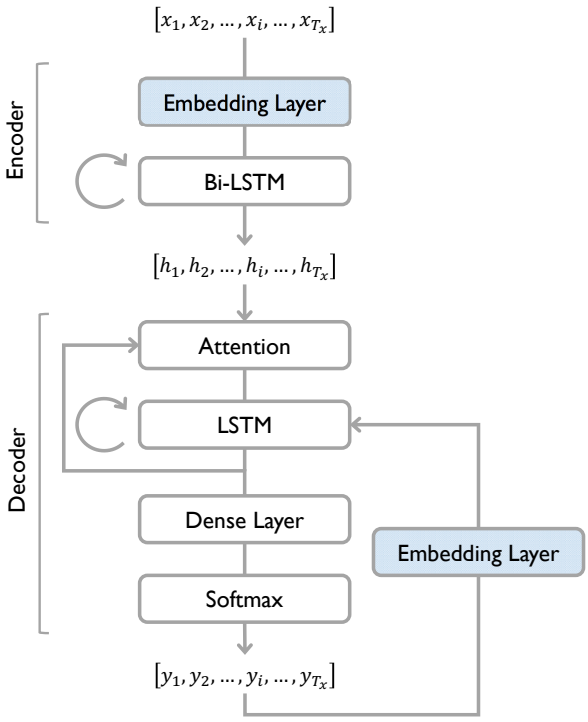

Figure 1: Basic structure of our system.

tional LSTM consists of a forward LSTM and a backward LSTM that move from left to right and right to left respectively to update their hidden states. The hidden states of the last layer are the outputs of the encoder which is then fed into the decoder.

Given the encoder output, the decoder generates an output sequence. Following Sutskever et al. (2014) and Bahdanau et al. (2015), we decided to use a multi-layer LSTM decoder with an attention mechanism. At each step, an attention mechanism computes a weighted average of vectors in the encoder output, called an attention context vector. A weight of a hidden state vector is computed using both itself and the hidden state of the decoder at that step. In addition to the attention context vector, the decoder also receives an embedding vector of the previous output token in order to retain the information of tokens it has already generated. These two vectors, an attention context vector and an embedding vector of the previous output token, are concatenated and given to the decoder LSTM, which then generates tokens and updates its state.

4 System Description

Our system implemented two tricks on a vanilla SEQ2SEQ model implemented by Google (Britz et al., 2017)[1] on Tensorflow[2] (ver. 1.0). The tricks are embedding layer initialization (§ 4.2) and batch size expansion (§ 4.3).

In what follows, we explain our system in detail. The basic structure of our system is depicted in Figure 1. The configuration and the default parameters used in our experiments are described in § 5.1 and in Appendix.

4.1 Preprocessing

As for the preprocessing, we basically followed the description of WAT 2017 Baseline Systems Data preparation.[3] We used scripts included in Moses toolkit[4] (ver. 2.2.1) (Koehn et al., 2007) for English tokenization and truecasing, and KyTea[5] (ver. 0.4.2) (Neubig et al., 2011) for Japanese segmentations.

After the above basic preprocessing, we applied SentencePiece,[6] which is an unsupervised text tokenizer and detokenizer, to the corpus. SentencePiece decides token boundaries using raw sentences (a white space is treated as a character) based on statistical models like character n-grams. This alleviates the problem of unknown tokens in a similar manner as using subword units. For this model, we picked unigram which is a default setting in the given implementation.

4.2 Embedding Layer Initialization

Because of the nature of the neural network model, each layer in the NMT model can only handle fixed-length inputs and outputs. Since our model is an end-to-end NMT model, both the first encoder layer and the decoder layer which feeds the previous output into the decoder accepts a vocabulary-size-length one-hot vector. In this regard, both layers are embedding layers which convert a one-hot vector into a word embedding vector.

Usually, all the layers, including embedding layers, are initialized randomly and trained in the exact same way. We attempted pretraining of these embedding layers, initializing them with word embeddings from an unsupervised neural language model trained on the training datasets in the source

[1] https://google.github.io/seq2seq/
[2] https://www.tensorflow.org/
[3] http://lotus.kuee.kyoto-u.ac.jp/WAT/WAT2017/baseline/dataPreparationJE.html
[4] http://www.statmt.org/moses/
[5] http://www.phontron.com/kytea/
[6] https://github.com/google/sentencepiece

and target languages. It is expected that these word embeddings improve the translation performance as well as speeding up convergence.

In addition to the original vocabulary, there are three special tokens in our system, "SEQUENCE_START," "SEQUENCE_END," and "UNK." The embeddings for the first two tokens were trained by adding them into the training dataset before the pretraining procedure. On the other hand, the embedding for "UNK" was generated by averaging all the out-of-vocabulary token embeddings.

Our proposed embedding layer initialization is a quick and simple trick, but effective on NMT systems (§ 5.3.1).

4.3 Using Large Batch Size

Gradient descent (GD) computes a gradient of parameters based on the entire dataset to update the parameters at each step. While this gives the most accurate gradient, it is computationally inefficient, as all data points need to be evaluated.

To overcome this issue, stochastic gradient descent (SGD) and its variants computes a gradient using a small portion of the dataset, called a mini-batch. We may consider the gradient computed in SGD as an expectation of the gradient which is inaccurate. However, as SGD is faster than GD, we can execute more steps which leads to better training in the same amount of time.

The accuracy of a gradient at each step depends on batch size, the number of samples in a mini-batch. A larger batch size leads to a more accurate gradient. The impact of large batch size on the translation quality will be investigated in § 5.3.2, in which we found that large batch size improves the translation significantly up to 256.

4.4 Ensemble

Ensemble of models is a widely used technique that improves the translation quality. After training several models, the decoders' outputs are combined to get the ensemble output. The effectiveness of ensemble was investigated in Denkowski and Neubig (2017).

For our system, we implemented a simple averaging ensemble. Let N be the number of models to ensemble, $X = \{x_1, x_2, \cdots, x_{T_x}\}$ and $Y = \{y_1, y_2, \cdots, y_{T_y}\}$ be the source and target sequences respectively, and $p_n(w|X, Y_{:j-1})$ be the probability of word w of the nth model at step j, where $Y_{:j-1}$ denotes the first $j-1$ tokens in the sequence Y. Then, the probability of word w is determined by taking the average of all models.

$$p(w|X, Y_{:j-1}) = \frac{1}{N}\sum_{n=1}^{N} p_n(w|X, Y_{:j-1})$$

Each model is independently trained in the training phase and the decoders' outputs are combined in the prediction phase. This simple technique gave us a significant BLEU score boost (§ 5.3.4).

4.5 Beam Search

Another technique to improve the translation quality is a beam search. The objective of translation system is

$$\hat{Y} = \arg\max_{Y \in \mathcal{Y}} p(Y|X)$$

where $\mathcal{Y}$ is the set of all possible translations and X is the input sequence. However, $\mathcal{Y}$ is such a huge set that computing $p(Y|X)$ for all $Y \in \mathcal{Y}$ is not realistic. A simple solution to this problem is to decide y_j to be

$$y_j = \arg\max_{w \in V} p(w|X, Y_{:j-1}).$$

This algorithm is called a greedy search. A greedy search algorithm is fast but may miss the best output sequence if the early portion of the sequence has a low probability.

The beam search algorithm addresses this issue by keeping multiple possible hypotheses, which are incomplete output sequences (Boulanger-Lewandowski et al., 2013). At each step, the top l hypotheses with the highest scores are kept for the next step. When every hypothesis terminates with an EOS token, the hypothesis with the highest score is chosen as the final result.

The beam search algorithm favors shorter sequences on average because a longer sequence tends to have a lower probability, $p(Y|X)$.

To overcome this problem, Wu et al. (2017) proposed a length penalty which gives advantages to longer sentences. With a length penalty, the score of a sequence Y given a source sequence X is computed by

$$score(Y|X) = \frac{\log(P(Y|X))}{lp(Y)}$$
$$lp(Y) = \frac{(5 + |Y|)^\alpha}{(5+1)^\alpha}$$

where α is a hyperparameter.

	Train		Dev		Test	
	en	ja	en	ja	en	ja
# sentences	1,783,817		1790		1812	
Ave. # tokens	31.08	33.13	31.06	34.58	30.69	34.03

Table 1: Details of corpus after preprocessing.

5 Evaluation

In this section, we report the default configuration of our system (§ 5.1) and the official evaluation result of our system for ASPEC English to Japanese translation subtask (§ 5.2). Furthermore, we report several other experiments that aim to show the effects of our tricks (§ 5.3).

5.1 Setup

The following settings are used as our default configuration in the experiments and the final system, unless otherwise noted. We use a two-layer bidirectional LSTM with dropout on input with $p = 0.8$ for the encoder, and a four-layer LSTM with the same dropout settings for the decoder. The number of units in hidden layers and the embedding dimension are set to 512. Adam (Kingma and Ba, 2015) is used for the optimizer, with a learning rate of 0.0001 and batch size is set to 256.

The vocabulary size after SentencePiece preprocessing is 16,000. The number of sentences and the average number of tokens after preprocessing in a single sentence are shown in Table 1.

As the default embedding method for embedding layer initializaion, we use Continuous Bag of Words (CBOW) (Mikolov et al., 2013) with window size of 5. We use word2vec (ver. 1.0)[7] with default parameters, except for the embedding dimension which was changed to 512. We train the word embeddings using only the preprocessed training dataset, in which both languages are concatenated to share the source and target vocabulary. All other layers were initialized randomly using uniform distribution.

We train the model for 200,000 steps, and at every 2000 steps during training, the current model is saved as a "checkpoint." When the training is done, all the checkpoints are evaluated using a greedy search algorithm on the development corpus. Only the checkpoint with the highest BLEU score is used for all of the following experiments and our final translation system. If the checkpoint with the highest BLEU score is at or near 200,000 steps (we define this as larger than 190,000 steps), we regard this model as not having converged, and will be identified as such in the results.

For all evaluations, KyTea segmentation was used to compute the BLEU score. For a prediction with the beam search algorithm, we used beam width of 128 except in our final system, which we used 256. For length penalty, we choose $\alpha = 1$ after parameter turning. Detailed settings are provided in the Appendix.

[7] https://github.com/svn2github/word2vec

ID	Hyperparameters			Dev		Test	
	batch size	hidden layer	learning rate	greedy	beam	greedy	beam
1	256	256	0.0001	34.22	35.65	34.40	35.54
2	256	384	0.0001	**35.32**	**36.74**	34.85	36.28
3	256	512	0.0001	35.22	36.48	34.81	36.29
4	256	512	0.0002	35.19	36.43	34.29	35.60
5	256	512	0.0005	34.40	36.08	34.19	35.57
6	256	768	0.0001	34.78	36.37	34.70	35.92
7	256	768	0.0002	34.97	36.46	**34.88**	**36.43**
8*	512	512	0.0001	34.62	36.61	34.68	36.35
9*	512	768	0.0001	34.31	36.42	34.28	35.97
10*	800	512	0.0001	30.05	33.73	29.35	33.36
Average				34.31	36.10	34.04	35.73
Best ensemble (2, 3, 4, 5, 6, 8, 9, 10)				38.00	39.03	37.40	38.93

Table 2: List of models trained for use in ensemble (* 200k steps unattained due to time constraints).

5.2 Official Evaluation Result

This section briefly explains how we built our final system and its result for the ASPEC English to Japanese translation subtask. We trained ten models with different hyperparameters which are listed in Table 2. For these models, we evaluated every possible ensemble combination using greedy search on the development corpus.[8] We chose the ensemble combination with the highest BLEU score to make prediction on the test corpus using a beam search algorithm. Consequently, we chose an ensemble of eight models, which achieved a BLEU score of 38.93 and an official human evaluation score of 68.000.

5.3 Further Investigations

In addition to the official evaluation, we conducted several other experiments. This section reports results and analyses of these experiments. We first confirm the impact of the embedding layer initialization, and then compare several word embedding methods (§ 5.3.1). Next, we investigate the

[8] Due to GPU memory limitations, three combinations are not evaluated: (2, 3, 4, 5, 6, 7, 8, 9, 10), (1, 3, 4, 5, 6, 7, 8, 9, 10), and (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

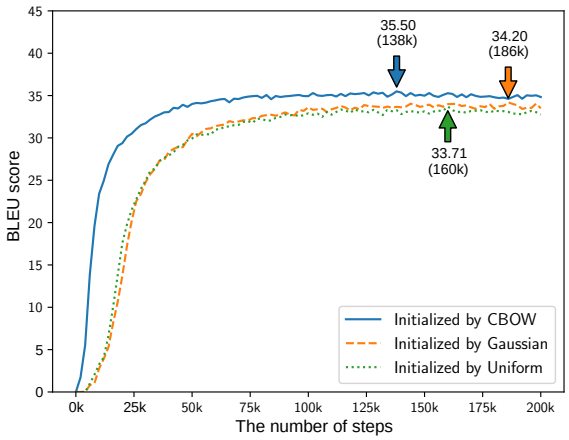

Figure 2: Training curve for models with different initialization methods. Arrows indicate the steps that achieved the best BLEU score on development data.

Initialization	Window	Greedy	Beam	Δ
Random (Gaussian)	-	34.20	35.57	-
Random (Uniform)	-	33.71	35.02	−0.55
CBOW	2	34.97	36.38	+0.81
	5	**35.50**	**36.85**	+1.28
	10	35.25	36.57	+1.00
Skip-gram	2	34.17	35.90	+0.33
	5	34.44	36.04	+0.47
	10	34.38	36.00	+0.43
SI-Skip-gram	2	34.04	35.16	−0.41
	5	34.44	35.91	+0.34
	10	34.33	35.69	+0.12
GloVe	2	34.50	36.01	+0.44
	5	34.58	35.86	+0.29
	10	33.98	35.39	−0.18
	15	34.35	36.00	+0.43

Table 3: Translation performance by embedding methods and window size. Evaluation is done on development dataset.

effect of batch size (§ 5.3.2). We then conduct experiments to discover the optimal learning rate when our initialization trick is employed (§ 5.3.3). Lastly, we examine the relation between the number of models used in the ensemble and translation performance (§ 5.3.4).

5.3.1 Impact of Embedding Layer Initialization

We first investigated the impact of our embedding layer initialization. The embeddings for the initialization are trained only on the training dataset of ASPEC using word2vec with CBOW and window size of 5. The question here is whether or not initialization with those word embeddings which were trained without any external data, by a task-independent, unsupervised method, improves the NMT model. In these experiments, the greedy search algorithm was used in order to obtain the training curve, as there are too many checkpoints to be evaluated by a beam search.

Figure 2 shows the training curve of three models, one initialized using CBOW, and the rest initialized randomly, with one using a Gaussian distribution, and the other a uniform distribution. The best score of the model with the CBOW initialization is 35.50 at step 138,000, and the best score of the model with random initialization is 34.20 with the Gaussian distribution at step 186,000. We observed that embedding layer initialization improves both the translation performance and the convergence time, increasing the former and decreasing the latter. Along with the following batch size experiment in § 5.3.2, the same experiment was done (using the greedy search algorithm) with batch sizes of 32, 64, 128, and 512, and this effect was observed across all batch sizes.

The results indicate that embedding layer initialization works in our NMT model, even though the embeddings are generated by CBOW, which is a totally task-irrelevant method.

Since we confirmed the effectiveness of our embedding layer initialization, we then investigate the effect of different embedding methods on translation performance. There are various methods other than CBOW to create word embeddings. Mikolov et al. (2013) proposed Skip-gram. Pennington et al. (2014) proposed another method called GloVe. Bojanowski et al. (2017) proposed Subword Information Skip-gram (SI-Skip-gram) that utilizes morphological information by including character n-grams of words in the model.

These methods train word embeddings using windows that obtain co-occurrences of neighboring words. It is known that a smaller window size leads to more syntactic embeddings and a larger one leads to more semantic embeddings (Lin and Wu, 2009; Levy and Goldberg, 2014).

The question is: which embedding method and window size yield the best results for the translation task when used to initialize the embedding layer? To answer this question, we trained 13 models using CBOW, Skip-gram, Subword Information Skip-gram (SI-Skip-gram), and GloVe, with window sizes of 2, 5, and 10, as well as a window size of 15 with GloVe, as this was its default value. For implementations of CBOW and Skip-

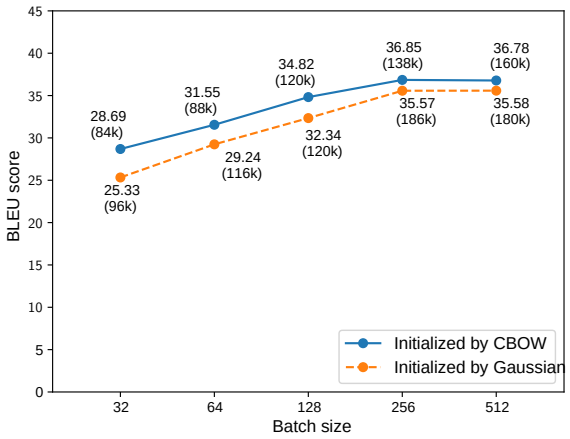

Figure 3: Translation performance by initialization and batch size. The values in the parentheses indicate the step that resulted in the best BLEU score on development data.

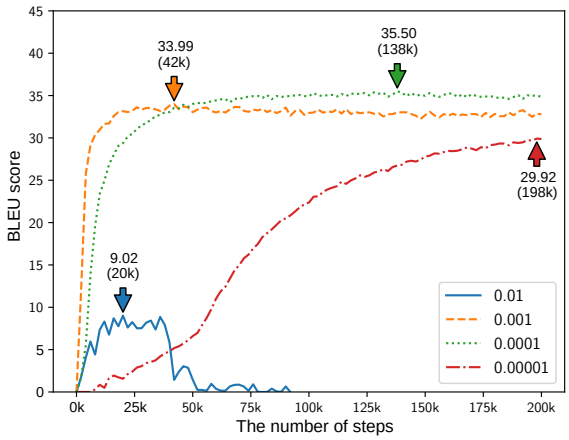

Figure 4: Training curve for different learning rates. Arrows indicate the steps that achieved the best BLEU score on development data.

gram, we used word2vec (ver. 1.0). For GloVe and SI-Skip-gram, we used GloVe (ver 1.2)[9] and fastText (ver. 1.0),[10] respectively.

The results are shown in Table 3. Most of the embedding methods outperformed random initialization by Gaussian distribution. This confirms the effectiveness of embedding layer initialization. Among those embedding methods, CBOW yields the best BLEU score of 35.50 for greedy search and 36.85 for beam search. For the window sizes, we found that each method has a different window size that yields the best result. Given this result, we decided to use CBOW with window size of 5 as our default setting.

5.3.2 Impact of Large Batch Size

We used the mini-batch method to train the network. While Morishita et al. (2017) investigated the effect of large batch size up to 64, it is unclear how an even larger batch size will impact translation performance. To evaluate this, we conducted experiments with different batch sizes.

Figure 3 confirms our idea and shows that, up until 256,[11] a larger batch size results in a better BLEU score, indicating that batch size has a significant impact on translation performance. The significance of the results are surprising, given the trick's simplicity.

When using this trick in NMT systems, it is important to recognize the tradeoff between translation performance and the memory and time required. In terms of the required memory, we were able to conduct the experiments up to a batch size of 256 on a server with 12GB of GPU memory, but a server with 24GB of GPU memory was required for experiments with a batch size of 512. Also, when we compared the time required to reach 200,000 steps when trained with batch sizes of 128 and 256, which were both trained on the same server, the larger batch size took 1.57 times as much time. The steps needed to reach the maximum BLEU score on development set became larger as the batch size increases, which indicates slower convergence with the larger batch size.

With the above factors taken into consideration, a batch size of 256 is a practical choice, and we can also expect an additive effect in translation quality by the use of CBOW initialization.

5.3.3 Impact of Learning Rate

An improperly large learning rate changes the values of each layers in a neural network drastically. Since we hypothesize that the pretrained embeddings have well-adjusted values, a drastic change in these values would spoil the effect of embedding layer initialization. To confirm this hypothesis, we compared four different learning rates of [0.01, 0.001, 0.0001, 0.00001] with the same configuration including initialization method.

Figure 4 shows the training curve of these four

[9] https://github.com/stanfordnlp/GloVe

[10] https://github.com/facebookresearch/fastText

[11] Initialized randomly by uniform distribution, the models achieved BLEU scores of 35.02 and 36.15 for batch sizes 256 and 512 respectively, and are seemingly still improving. However, we think this is within fluctuation range caused by random initialization.

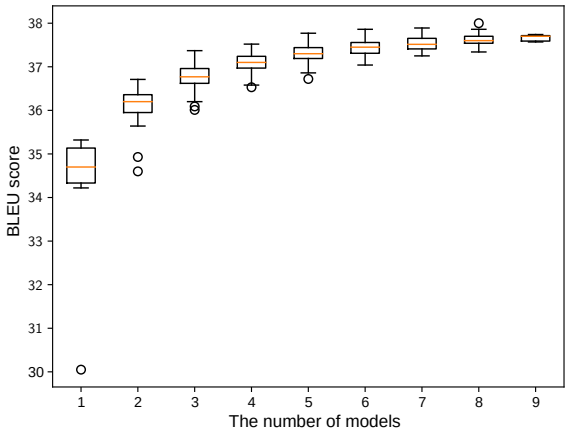

Figure 5: Performance of ensemble. As a beam search is costly, a greedy search was used.

different models. A learning rate of 0.01 performed abysmally, not only in terms of the worst best score but also in terms of the unstable training curve. The neural network could not be successfully trained at this learning rate. With the learning rate less than or equal to 0.001, the training curve becomes stable and the best score marks a reasonable value. A learning rate of 0.001 resulted in the best score of 33.99 at step 42,000, which is good and fast enough. As expected, a learning rate of 0.0001 raised the best score to 35.50 at step 138,000, which is 1.51 higher than the score with a learning rate of 0.001, but at a much later step. The best score for learning rate of 0.00001 was 29.92 at step 198,000, but the model did not converge.

It is difficult to confirm our hypothesis that smaller learning rate is always better for keeping the well-adjusted values by the initialization, with the above results. However, considering the fact that the time spent on training is limited, we believe 0.0001 to be the most practical learning rate among them, because it marked a score almost 1.0 higher than the second best one.

5.3.4 Ensemble Strategy

It is known that ensemble technique improves translation (Denkowski and Neubig, 2017). The intuition is that the larger the number of models is, the better the translation will be. To test this hypothesis, we exhaustively compared the results of ensembles with a different number of models.

The ten models from Table 2 were used for this experiment. We evaluated ensembles of all possible combinations. As mentioned in the footnote in § 5.2, three combinations are omitted because of the memory limitation, which yielded 1,020 combinations in total.

The result is reported in Figure 5. We can see the positive correlation between number of models used in the ensemble and the performance. However, as the number of models gets bigger, the effect of adding models gets smaller; the difference between a single model and two model ensemble is significant, but the difference between an eight model ensemble and a nine model ensemble is not so evident.

6 Conclusion

We have described the translation system, experiments, and the results of the team UT-IIS. As for the result of our system on the ASPEC En-Ja task, we were able to achieve a BLEU score of 38.93, which is higher than the score for the state-of-the-art system of WAT 2016. This reflects the effectiveness of our word embedding layer initialization technique, when combined with model ensemble and a beam search on the vanilla SEQ2SEQ model. Our findings are as follows:

- Embedding layer initialization technique using only the parallel corpus improves translation quality (§ 5.3.1).
- Embedding layer initialization trick with CBOW works the best (§ 5.3.1).
- Benefits of a larger batch size reached saturation at 256, and we believe this to be the practical setting (§ 5.3.2).
- A learning rate of 0.0001 is both good and fast enough to be practical with the initialization trick (§ 5.3.3).
- Ensemble of many models improves translation quality significantly (§ 5.3.4).

We believe that the embedding layer initialization technique, as well as the insights gained from our experiments, will contribute to the improvement of NMT when used in combination with other novel techniques.

We have published our code on `https://github.com/nem6ishi/wat17`.

Acknowledgements This work was partially supported by JSPS KAKENHI Grant Number 16K16109 and 16H02905.

model	AttentionSeq2Seq
model_params	
attention.class	seq2seq.decoders.attention.AttentionLayerBahdanau
attention.params	
num_units	512
bridge.class	seq2seq.models.bridges.ZeroBridge
embedding.dim	512
encoder.class	seq2seq.encoders.BidirectionalRNNEncoder
encoder.params	
rnn_cell	
cell_class	LSTMCell
cell_params	
num_units	512
dropout_input_keep_prob	0.8
dropout_output_keep_prob	1.0
num_layers	2
decoder.class	seq2seq.decoders.AttentionDecoder
decoder.params	
rnn_cell	
cell_class	LSTMCell
cell_params	
num_units	512
dropout_input_keep_prob	0.8
dropout_output_keep_prob	1.0
num_layers	4
optimizer.name	Adam
optimizer.params	
epsilon	0.0000008
optimizer.learning_rate	0.0001
source.max_seq_len	50
source.reverse	false
target.max_seq_len	50

Table 4: Configuration of seq2seq model.

A Hyperparameters and configuration

Table 4 lists the default hyperparameters and configuration for our system, which is built based on Google's implementation of the SEQ2SEQ model (Britz et al., 2017).

References

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), Short Papers*, pages 132–140.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the Third International Conference on Learning Representations (ICLR)*.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems (NIPS) 19*, pages 153–160.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association of Computational Linguistics*, 5:135–146.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. 2013. Audio chord recognition with recurrent neural networks. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, pages 335–340.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv: 1703.03906*.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017. Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1936–1945.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Fabien Cromieres, Chenhui Chu, Toshiaki Nakazawa, and Sadao Kurohashi. 2016. Kyoto university participation to WAT 2016. In *Proceedings of the Third Workshop on Asian Translation (WAT)*, pages 166–174.

Michael Denkowski and Graham Neubig. 2017. Stronger baselines for trustable results in neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation (WNMT)*, pages 18–27.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 823–833.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), Short Papers*, pages 72–78.

Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin. 2017. A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 123–135.

Çaglar Gülçehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loïc Barrault, Huei-Chi Lin, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2015. On using monolingual corpora in neural machine translation. *arXiv preprint arXiv: 1503.03535*.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Shonosuke Ishiwatari, Jingtao Yao, Shujie Liu, Mu Li, Ming Zhou, Naoki Yoshinaga, Masaru Kitsuregawa, and Weijia Jia. 2017. Chunk-based decoder for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1901–1912.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1700–1709.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On large-batch training for deep learning: Generalization gap and sharp minima. In *Proceedings of 5th International Conference on Learning Representations (ICLR)*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the third International Conference on Learning Representations (ICLR)*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL), Demo and Poster Sessions*, pages 177–180.

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL), Short Papers*, pages 302–308.

Junhui Li, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. 2017. Modeling source syntax for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 688–697.

Dekang Lin and Xiaoyun Wu. 2009. Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 1030–1038.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of the First International Conference on Learning Representations (ICLR)*.

Makoto Morishita, Yusuke Oda, Graham Neubig, Koichiro Yoshino, Katsuhito Sudoh, and Satoshi Nakamura. 2017. An empirical study of mini-batch creation strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 61–68.

Toshiaki Nakazawa, Shohei Higashiyama, Chenchen Ding, Hideya Mino, Isao Goto, Graham Neubig, Hideto Kazawa, Yusuke Oda, Jun Harashima, and Sadao Kurohashi. 2017. Overview of the 4th Workshop on Asian Translation. In *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, Taipei, Taiwan.

Toshiaki Nakazawa, Manabu Yaguchi, Kiyotaka Uchimoto, Masao Utiyama, Eiichiro Sumita, Sadao Kurohashi, and Hitoshi Isahara. 2016. ASPEC: Asian scientific paper excerpt corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 2204–2208.

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT), Short Papers*, pages 529–533.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Prajit Ramachandran, Peter Liu, and Quoc Le. 2017. Unsupervised pretraining for sequence to sequence learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 383–391.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 379–389.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1715–1725.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS) 27*, pages 3104–3112.

Subhashini Venugopalan, Lisa Anne Hendricks, Raymond Mooney, and Kate Saenko. 2016. Improving LSTM-based video description with linguistic knowledge mined from text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1961–1966.

Shuangzhi Wu, Dongdong Zhang, Nan Yang, Mu Li, and Ming Zhou. 2017. Sequence-to-dependency neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 698–707.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv: 1609.08144*.

Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. 2016. Transfer learning for low-resource neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1568–1575.